Comparing Genetic Algorithm and Particle Swarm Optimization for optimization of Neural Nets

Kabilan V
19BCE0623
School Of Computer Science and Engineering
VIT Vellore
kabilan.v2019@vitstudent.ac.in

Devesh D R
19BCE0674
School Of Computer Science and Engineering
VIT Vellore
deveshd.r2019@vitstudent.ac.in

Praveen Raj M
19BCE0685
School Of Computer Science and Engineering
VIT Vellore
praveenraj.m2019@vitstudent.ac.in

G Aadhithyaa
19BCI0121
School Of Computer Science and Engineering
VIT Vellore
aadhithyaag.2019@vitstudent.ac.in

***Abstract*— Neural Nets are one of the fastest growing aspects of AI with a wide and versatile range of applications. For these models to be used in various applications it is required to be accurate. To achieve this, it must have suitable hyper parameters and weights to train with. These hyper parameters are the main problem and need to be found out. There are no perfect set of hyper parameters for a particular application. The technique of finding the right hyper parameters works by means of trial and error. In this work we try to apply Meta heuristic algorithms such as Genetic Algorithm and Particle Swarm Optimization algorithm to find the optimum set of hyper parameters for training a Neural Net model that can give best results.**

This work also compares the optimization algorithms with various factors like computation time, time to convergence and tries to find the best one.

Keywords—PSO, GA, ANNs, CNNs,AI, EVOLUTIONARY ALGORITHMS

I. Introduction

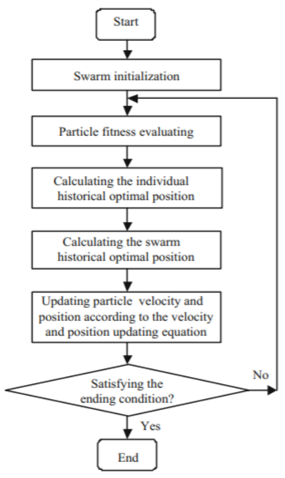
Evolutionary Methods such as genetic algorithms and Particle swarm optimization have proven to be highly effective in solving a wide range of problems, as they can be used in realistic decision-contexts. This is because they can be used as one element in broader, participatory environmental decision-making processes. They can be linked with existing simulation models to assist with the exploration of large solution spaces. Generally provide a number of "good" solutions that can be explored further, rather than being prescriptive. These algorithms can solve multi-objective problems and can take uncertainties(constraints) into account.

A. Particle Swarm Optimization

First, Particle swarm optimization (PSO) is a population-based optimization technique inspired by the motion of bird flocks and schooling fish. In PSO, the potential solutions, called particles, move in the problem space by following the current optimum particles. The system is initialized with a population of random solutions, and the search for the optimal solution is performed by updating generations. This is done by using velocity and distance as important components. Figure # explains the flow of the algorithm. The algorithm starts with a random initial swarm. The fitness value is evaluated for this initial swarm. The individual historical optimal position and swarm's historical optimal position is calculated. After this each particle's velocity and distance is updated using the formulas from Fig 1. This happens until the stopping criteria is met. PSO is computationally more efficient in terms of both speed and memory requirements.

$$v_i = Wv_i + c_1r_1(P_{best,i} - x_i) + c_2r_2(g_{best} - x_i)$$
$$x_i = x_i + v_i$$

v_i	Velocity of the ith particle	$P_{best,i}$	Personal best of the ith particle
x_i	Position of the ith particle	g_{best}	Global best
r_1 and r_2	Random numbers	c_1 and c_2	Acceleration Coefficients
W	Inertia weight		



Start
Swarm initialization
Particle fitness evaluating
Calculating the individual historical optimal position
Calculating the swarm historical optimal position
Updating particle velocity and position according to the velocity and position updating equation
Satisfying the ending condition?
No
Yes
End

Fig 1. Diagram representing flowchart for PSO

According to the paper by M. Clerc and J. Kennedy, to define a standard for Particle Swarm Optimization, the best static parameters are w=0.72984 and c1 + c2 > 4. More exactly c1 = c2 = 2.05. So we used the same values for our hyperparameters in our work.

B. Genetic Algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. GAs were developed by John Holland and his students and colleagues at the University of Michigan, Genetic Algorithms have the ability to deliver a "good-enough" solution "fast-enough". This makes genetic algorithms attractive for use in solving optimization problems. The flow of the algorithm is shown in Fig. 2

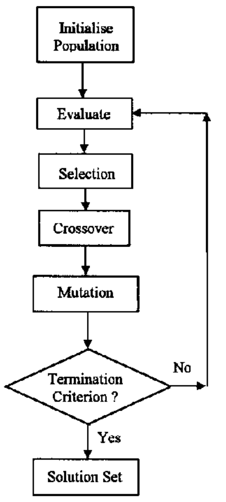


Fig 2. Diagram representing flowchart for Genetic Algorithm

Genetic Algorithm begins by initializing a population of all possible values given from the list. The range of values from which to select is given and the algorithm initializes the population. This population is evaluated using the fitness function. Based on the performance measure the algorithm selects from the members of the population most likely to give the best results according to the performance measure. From these members the best characteristics are chosen and used to create offspring in the crossover function. Based on random probability these offspring undergo mutation to explore previously unexplored values in the given range to find all the rest of the values and not only be limited to the values the first population was initialized with. Once this is done the performance measure is taken and tested to find if the stopping criteria is reached. If the criteria is not yet met the algorithm loops back to the evaluation function and repeats the optimization process until the given criteria is met. Once the best possible values are taken from the range of the population the algorithm exits and returns a solution set.

II. OBJECTIVE

Current objectives for this project include understanding Neural Networks in depth and potential usage. We train Neural Networks using potential dataset and analyse the metrics of the network (error, accuracy). We identify suitable hyper parameters and weights to train with by applying trial and error processes. We apply Meta Heuristic Algorithms such as Genetic Algorithm and Particle Swarm Optimization to identify the accurate weights and hyper parameters. The comparison of performance was mainly based on three parameters. The number of iterations needed to converge, CPU utilization time and the quality of solutions.

A. Applications

Define Neural Networks have a huge range of applications in various different fields. Neural Networks have been extensively used in the field of image processing and pattern recognition systems. One of its most common use cases is Facial Recognition Systems, which are used in state of the art authentication and surveillance systems. Neural Networks are used to train recognition systems from a large database of images and that is used to give accurate recognition results. Another major application of neural networks is forecasting. Neural Networks are used to analyse a stock's previous performance, its returns and profit ratios and is used to predict the volatility of the market and forecast stock market prices. Neural networks are also used for signature verification, handwriting analysis, weather forecasting and can be used in modelling non-linear dynamical systems. Extensive research is still ongoing for the use of Neural Networks in various other fields.

Multiple optimization methods are used to optimize the performance of neural networks . Optimization varies the attributes of neural networks such as weights, hyperparameters and learning rate to reduce losses. Optimization mainly helps to achieve more accurate results from neural networks, which can aid the applications mentioned above by a great deal.

B. Limitations

There are a few limitations to the use of Genetic Algorithms and Particle Swarm Optimization. The Genetic Algorithm does not require a lot of information about the problem and hence it can be very difficult to design and implement an objective fitness function. Genetic Algorithms have limited use cases due to its shortcomings such as an unclear stop criteria and local optimum traps. Above all this, using a genetic algorithm is also computationally extensive. Particle Swarm Optimization has the risk of a low convergence rate. From previous works it is also observed that PSO has low quality results. Also, PSO takes up more memory than GA in order to store and update velocity.

Since both the algorithms have their part of advantages and disadvantages, we planned to explore both the algorithms and find the best one.

III. SYSTEM ANALYSIS

A. *Existing System*

The existing system which is an artificial neural network is a computing system that uses a system of highly interconnected nodes or neurons organized in layers to process inputs and use dynamic states to return outputs. The complex patterns present in data and information can be found out using these nodes and even patterns that are too complex for being manually extracted and taught to recognize by machines. This current neural network uses an input layer that has one neuron for each component present in the input data and communicates this to a number of hidden layers. These hidden layers are not part of the input or output layers. The system uses weights and biases to calculate weights and moves to output to give the result.

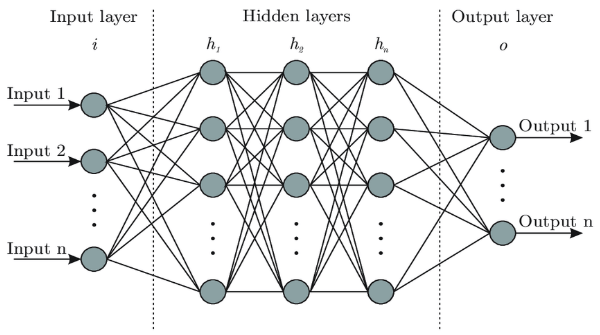


Fig 3. Existing System Diagrammatic Representation

From the figure we can see that the input is received, the neurons calculate a weighted sum adding also the bias according to the result and using a specific activation function. The neuron passes this information to successive hidden layers in a process called a forward pass. Finally the last hidden layer is connected to the output layer which has one neuron for each possible class that the neural network should give output as and finally gives the result.

B. *Proposed System*

Normally the hyper parameters such as the learning rate, number of hidden layers and number of neurons and so on are hardcoded and the weights and biases undergo training to reduce the error. Our Proposed system works by initializing hyperparameters and population. These are a set of random sets of hyperparameters that the neural network can use. The best possible combination of these hyperparameters are not known and are all entered as part of the population. This is then randomly initialized and passed onto the neural network for optimization. The performance of the neural network is noted and then used to evaluate the Genetic Algorithm or Particle Swarm Optimization function. Using the performance measure we determine if the stopping criteria is met. If the stopping criteria is not met we use the GA and PSO to optimize the hyper parameters and select them to use in the Artificial neural network and the performance measure is taken again. This is repeated until the stopping criteria is met. Once the stopping criteria is met the end of optimization and we find the best possible set of hyperparameters to get the most accuracy from. We use these results to compare GA and PSO and evaluate both the algorithms.

C. *Benefits of Proposed System*

Machine learning algorithms are being used to deal with a variety of practical problems like computer vision and speech processing. But the performance of these machine learning algorithms being used is largely determined by their hyper-parameters and without good hyper-parameter values the performance of these algorithms will be very poor. Unfortunately, for such deep learning models like these artificial neural networks, it is very difficult to determine their hyper-parameters. Thus developing an efficient algorithm for hyper-parameters to be determined automatically will be of important significance. The trial and error method can be avoided and the model will not need new hyper-parameters to be manually entered by the programmer. They can be automatically determined and not only does it increase the chance of getting more optimized and getting a better combination it also can try out a larger number of combinations of hyper-parameters optimization which would have been missed had they been entered manually.

IV. REQUIREMENT SPECIFICATION

A. *Hardware Requirements*

Hardware is used by Google Collaboratory from its own Virtual machines. The specifications used by the Collaboratory VM are an Nvidia K80/Tf GPU with a GPU memory 12GB/16GB with a Memory Clock 0.82GHz / 1.59GHz capable of a performance of 4.1 TFLOPS / 8.1 TFLOPS. The VM also consisted of 2 CPU cores, 12GB RAM and a disk space of 358GB.

B. *Software Requirements*

The programs were coded using Python 3.9 using various libraries that were inbuilt including Tensorflow, Scikit learn, Matplotlib, Keras, Pandas and Numpy.

V. SYSTEM DESIGN SPECIFICATION

The main aim of the project is to improve the performance of the neural net and accuracy is selected as the target parameter for the purpose. So the system should be designed in the way it fulfills this purpose. The hyper parameters that are selected to optimize the system are the depth (number of layers), number of neurons in each layer, activation function, optimizer.

A. *System Architecture*

As discussed to apply genetic algorithm and particle swarm optimization on neural networks and optimize accuracy based on hyperparameters we need a system architecture that can train the neural network, calculate accuracy, tune hyperparameters based on the accuracy and continue this process until stopping criteria is met (total number of iterations or target accuracy). Then we return the best hyperparameters as the output. The detailed diagram of the architecture is discussed in the next section.

B. *Detailed Design*

So the initial hyper parameters for the population are selected randomly and the neural network is trained based on those hyperparameters, now accuracy is calculated and stopping criteria is checked. So if the stopping criteria is met the algorithm stops. Else the results are passed to the algorithms (Genetic algorithm or particle swarm optimization respectively). This training and calculating accuracy module acts as the fitness function for the algorithms. Now as discussed in the introduction, the algorithms tune the hyperparameters based on the results and give a new set of population or swarm for next iteration till the stopping criteria is met.

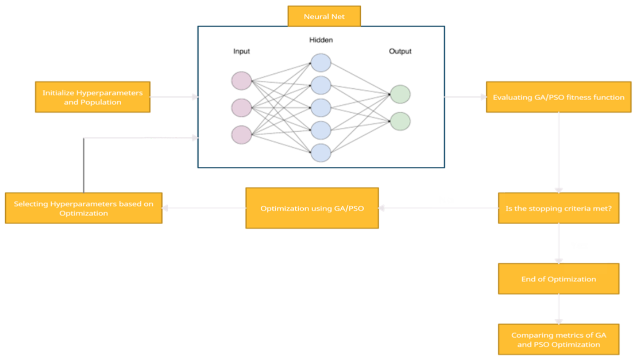

Neural Net
Hidden
Input
Output
Initialize Hyperparameters and Population
Evaluating GA/PSO fitness function
Selecting Hyperparameters based on Optimization
Optimization using GA/PSO
Is the stopping criteria met?
End of Optimization
Comparing metrics of GA and PSO Optimization

Fig 4. Detailed flowchart for our proposed system

C. Dataset

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It shares the same image size and structure of training and testing splits. It is intended that Fashion-MNIST will serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. Being a vast dataset with minimalist properties makes this dataset an excellent choice for exploration and comparison of these algorithms.

VI. System Implementation

The comparison of performance was mainly based on three parameters. The number of iterations needed to converge, CPU utilization time and the quality of solutions. To do this we ran the algorithm for different iterations and for different population or swarm size respectively.

A. Results

Swarm Size	Iterations	Parameters	Accuracy	Time Taken	Iteration to converge
20	10	[3, [64, 'selu'], [32, 'hard_sigmoid'], 'selu', ['categorical_hinge', 'adamax']]	77.60	14m	4
20	20	[5, [64, 'selu'], [32, 'linear'], [16, 'selu'], [32, 'hard_sigmoid'], 'selu', ['categorical_hinge', 'adamax']]	80.40	31m	14
20	25	[8, [128, 'hard_sigmoid'], [32, 'softsign'], [256, 'softsign'], [32, 'linear'], [16, 'selu'], [64, 'selu'], [256, 'softsign'], 'tanh', ['mean_absolute_error', 'adamax']]	80.40	43m	15
20	30	[11, [128, 'hard_sigmoid'], [32, 'softsign'], [256, 'softsign'], [64, 'tanh'], [32, 'linear'], [256, 'hard_sigmoid'], [256, 'hard_sigmoid'], [512, 'softmax'], [256, 'softsign'], [16, 'tanh'], 'hard_sigmoid', ['binary_crossentropy', 'nadam']]	79.19	55m	21
20	40	[10, [128, 'hard_sigmoid'], [32, 'softsign'], [256, 'softsign'], [128, 'elu'], [64, 'selu'], [16, 'softplus'], [128, 'elu'], [256, 'softsign'], [32, 'hard_sigmoid'], 'selu', ['categorical_hinge', 'adamax']]	80.75	1hr 9m	29

Table 1: Particle Swarm Optimization Observation - Varying Iterations

Swarm Size	Iterations	Parameters	Accuracy	Time Taken	Iteration to converge
10	20	[6, [32, 'softsign'], [128, 'elu'], [32, 'softplus'], [128, 'hard_sigmoid'], [512, 'softmax'], 'linear', ['mean_squared_logarithmic_error', 'adamax']]	81.80	11m	7
15	20	[5, [64, 'selu'], [512, 'linear'], [128, 'softsign'], [128, 'linear'], 'softsign', ['binary_crossentropy', 'sgd']]	82.78	21m	9
25	20	[6, [32, 'softsign'], [16, 'selu'], [32, 'softsign'], [256, 'sigmoid'], [256, 'softsign'], 'tanh', ['mean_absolute_error', 'adamax']]	82.68	36m	10
30	20	[7, [32, 'softsign'], [128, 'elu'], [32, 'sigmoid'], [256, 'sigmoid'], [512, 'softmax'], [32, 'hard_sigmoid'], 'selu', ['categorical_hinge', 'adamax']]	83.81	43m	12
40	20	[7, [64, 'selu'], [32, 'linear'], [16, 'selu'], [256, 'relu'], [64, 'hard_sigmoid'], [128, 'tanh'], 'relu', ['squared_hinge', 'rmsprop']]	83.76	59m	15

Table 2: Particle Swarm Optimization Observation - Varying Swarm Size

Population Size	Generations	Parameters	Accuracy	Time Taken	Iteration to converge
20	10	[3, [64, 'tanh'], [512, 'selu'], 'softmax', ['categorical_hinge', 'adamax']]	68.03	19m	6
20	20	[8, [64, 'selu'], [256, 'softsign'], [64, 'tanh'], [32, 'linear'], [32, 'linear'], [16, 'selu'], [32, 'hard_sigmoid'], 'selu', ['categorical_hinge', 'adamax']]	71.03	37m	17
20	25	[10, [64, 'softsign'], [16, 'linear'], [32, 'tanh'], [512, 'selu'], [64, 'softsign'], [16, 'linear'], [16, 'linear'], [1024, 'softsign'], [128, 'linear'], 'softmax', ['hinge', 'nadam']]	75.12	52m	21
20	30	[3, [1024, 'softsign'], [32, 'hard_sigmoid'], 'softmax', ['squared_hinge', 'adam']]	77.38	1h 8m	27
20	40	[4, [1024, 'selu'], [256, 'elu'], [512, 'softsign'], 'tanh', ['mean_squared_error', 'sgd']]	85.42	1h31m	35

Table:3 Genetic Algorithm Observation - Varying Iterations

Population Size	Generations	Parameters	Accuracy	Time Taken	Iteration to converge
10	20	[3, [256, 'sigmoid'], [1024, 'softplus'], 'softsign', ['squared_hinge', 'rmsprop']]	67.94	17m	12
15	20	[5, [64, 'softsign'], [128, 'selu'], [128, 'selu'], [512, 'linear'], 'sigmoid', ['poisson', 'rmsprop']]	72.21	28m	14
25	20	[6, [128, 'softsign'], [512, 'softsign'], [512, 'softsign'], [512, 'softsign'], [1024, 'softmax'], 'softplus', ['categorical_hinge', 'rmsprop']]	74.04	44m	15
30	20	[8, [512, 'sigmoid'], [256, 'relu'], [32, 'softplus'], [64, 'tanh'], [32, 'softplus'], [64, 'tanh'], [32, 'linear'], 'softmax', ['poisson', 'adam']]	77.54	57m	17
40	20	[9, [128, 'softsign'], [256, 'selu'], [64, 'selu'], [128, 'softsign'], [64, 'softsign'], [16, 'tanh'], [64, 'selu'], [64, 'tanh'], 'softmax', ['squared_hinge', 'adam']]	78.01	1h13m	19

Table 4: Genetic Algorithm Observation - Varying Population Size

B. Comparison of Results

Comparing Algorithms

Swarm Size	Iterations	Time Taken
20	10	14m
20	20	31m
20	25	43m
20	30	55m
20	40	1hr 9m

Swarm Size	Iterations	Time Taken
10	20	11m
15	20	21m
25	20	36m
30	20	43m
40	20	59m

Table 5: Particle Swarm Optimization - CPU Utilization Time

Population Size	Iterations	Time Taken
20	10	19min
20	20	37m
20	25	52 min
20	30	1 hr 8 min
20	40	1 hr 31min

Population Size	Generations	Time Taken
10	20	17m
15	20	28m
25	20	44m
30	20	57m
40	20	1h13m

Table 6: Genetic Algorithm - CPU Utilization Time

Swarm Size	Iterations	Iteration to converge
20	10	4
20	20	14
20	25	15
20	30	21
20	40	29

Swarm Size	Iterations	Iteration to converge
10	20	7
15	20	9
25	20	10
30	20	12
40	20	15

Table 7: Particle Swarm Optimization - Iterations for convergence

Population Size	Iterations	Iteration to converge
20	10	6
20	20	17
20	25	21
20	30	27
20	40	35

Population Size	Iterations	Iteration to converge
10	20	12
15	20	14
25	20	15
30	20	17
40	20	19

Table 8: Genetic Algorithm - Iterations for convergence

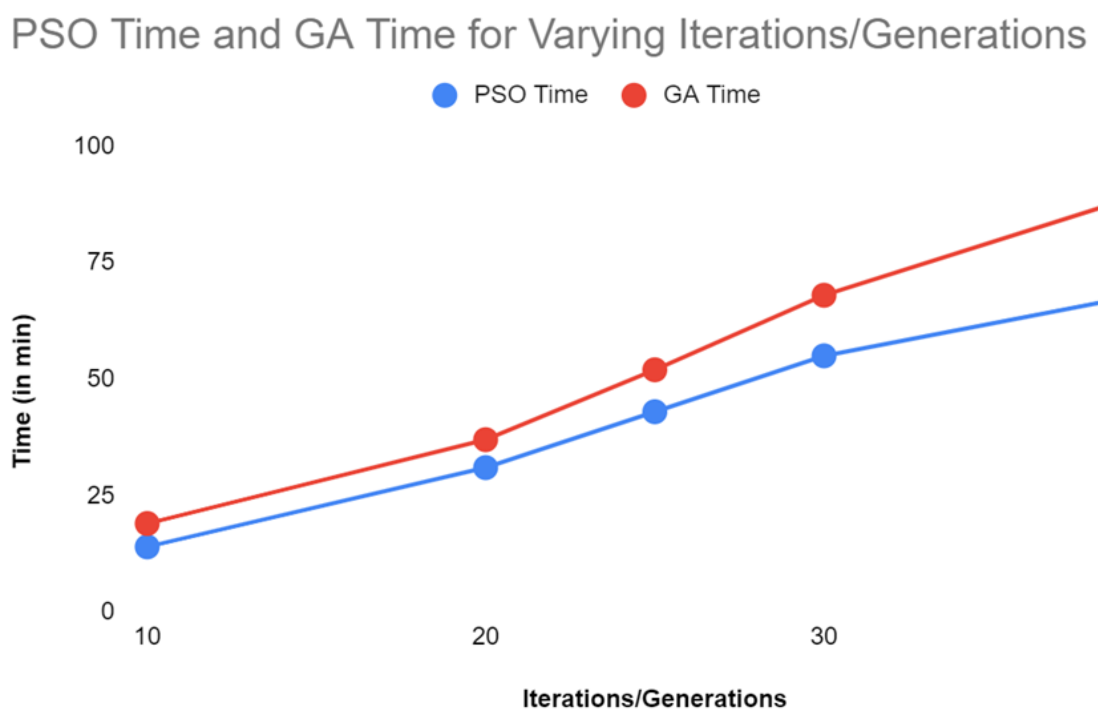


Fig 5. Graph showing the time taken by both PSO and GA for varying Iterations/Generations

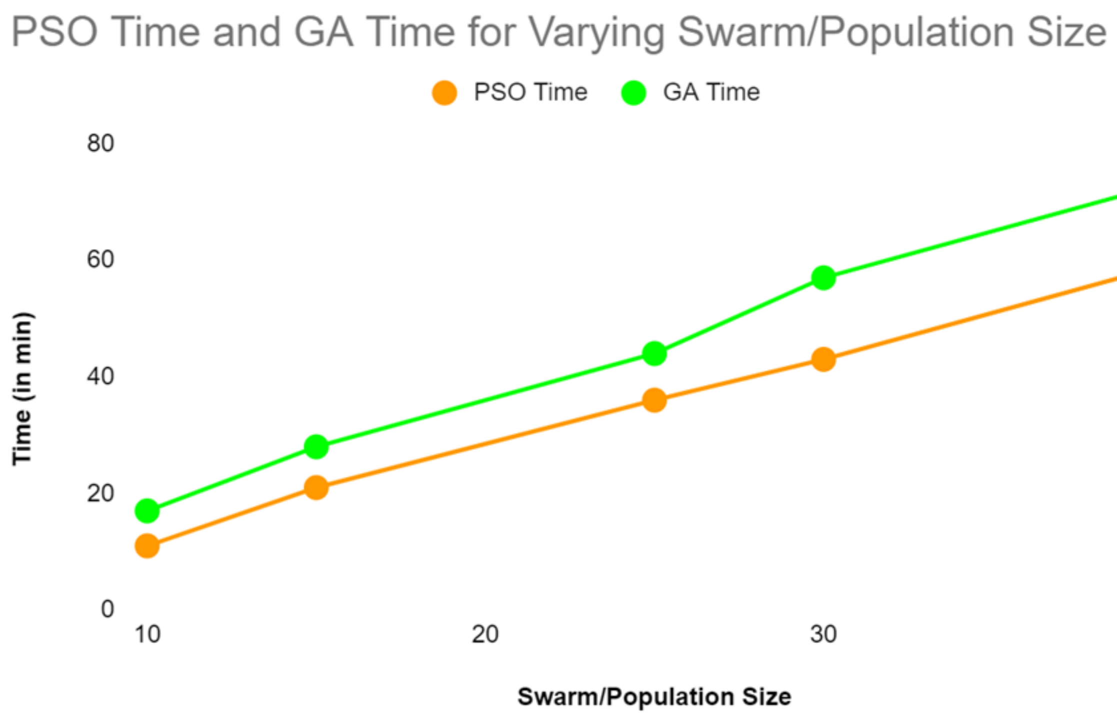


Fig 6. Graph showing the time taken by both PSO and GA for varying Swarm/Population Size

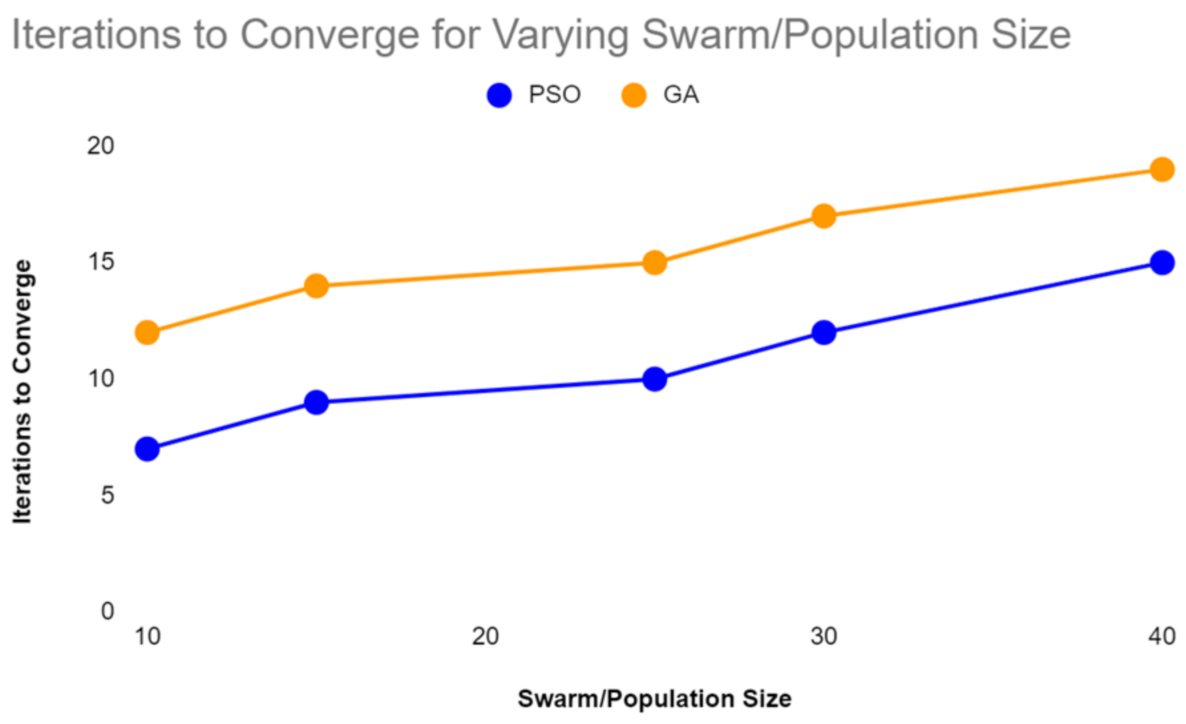


Fig 7. Graph showing the Iterations to Converge for both PSO and GA for varying Swarm/Population Size

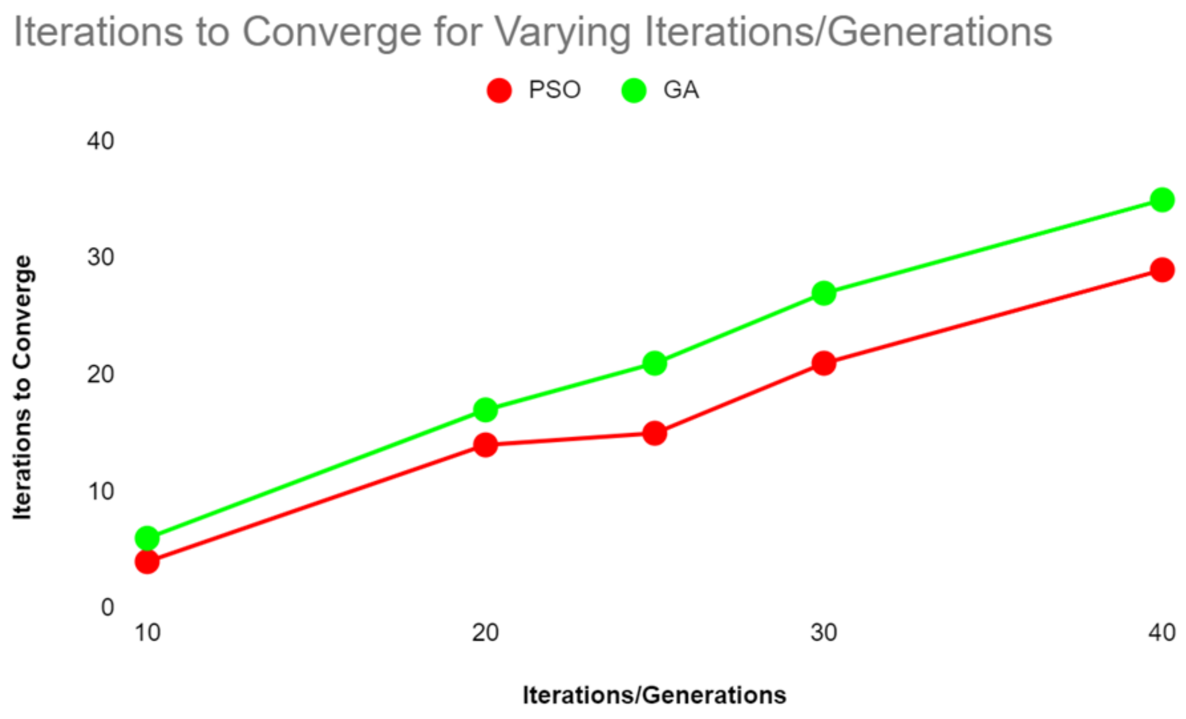


Fig 8. Graph showing the Iterations to Converge for both PSO and GA for varying Iterations/Generations

A. Observations

Particle swarm optimization (PSO) and genetic algorithm (GA) are both nature-inspired population based optimization methods. Compared to GA, whose long history can trace back to 1975, PSO is a relatively new heuristic search method first proposed in 1995.

Unlike GA, the variables in PSO can take any values based on their current position in the particle space and the corresponding velocity vector. PSO yields better quality and faster performance relative to GA. GA usually converges towards a local optimum or even arbitrary points rather than the global optimum of the problem while PSO tries to find the global optima.

VII CONCLUSION AND FUTURE ENHANCEMENTS

Our work concludes that Particle swarm optimization performs better than genetic algorithms in most of the comparing criteria. However Genetic algorithms prove to give better results if initial exploration is good. Training neural networks and improving their performance by tuning the hyper parameters is done by both the algorithms acceptably.

The real world datasets are quite larger than the fashion mnist dataset used in our work. It is a standard dataset capable of being used as a benchmark. So the computational time will drastically increase if the size or complexity of the dataset is increased. So as future work, we suggest implementing a parallel version of these meta-heuristic algorithms which can deal with this computational time significantly and can be used in real world datasets with powerful systems. Even though advantages and disadvantages of GA and PSO are explored in our project, optimization of neural networks are not limited only to these two algorithms.

References

[1] ul Islam, B., Baharudin, Z., Raza, M.Q. and Nallagownden, P., 2014, June. Optimization of neural network architecture using genetic algorithms for load forecasting. In 2014 5th International Conference on Intelligent and Advanced Systems (ICIAS) (pp. 1-6). IEEE.

[2] Chiroma, H., Abdulkareem, S., Abubakar, A. and Herawan, T., 2017. Neural networks optimization through genetic algorithm searches: a review. Appl. Math. Inf. Sci, 11(6), pp.1543-156.

[3] Gudise, V.G. and Venayagamoorthy, G.K., 2003, April. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706) (pp. 110-117). IEEE.

[4] Lydia, A. and Francis, F.S., A Survey of Optimization Techniques for Deep Learning Networks.

[5] Gudise, V.G. and Venayagamoorthy, G.K., 2003, April. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706) (pp. 110-117). IEEE.

[6] Guo, B., Hu, J., Wu, W., Peng, Q. and Wu, F., 2019. The Tabu_genetic algorithm: a novel method for hyper-parameter optimization of learning algorithms. Electronics, 8(5), p.579.

[7] Lorenzo, P.R., Nalepa, J., Kawulok, M., Ramos, L.S. and Pastor, J.R., 2017, July. Particle swarm optimization for hyper-parameter selection in deep neural networks. In Proceedings of the genetic and evolutionary computation conference (pp. 481-488).

[8] Guo, Y., Li, J.Y. and Zhan, Z.H., 2020. Efficient hyperparameter optimization for convolution neural networks in deep learning: A distributed particle swarm optimization approach. Cybernetics and Systems, 52(1), pp.36-57.

[9] Han, J.H., Choi, D.J., Park, S.U. and Hong, S.K., 2020. Hyperparameter optimization using a genetic algorithm considering verification time in a convolutional neural network. Journal of Electrical Engineering & Technology, 15(2), pp.721-726.

[10] Wicaksono, A.S. and Supianto, A.A., 2018. Hyper parameter optimization using genetic algorithms on machine learning methods for online news popularity prediction. International Journal of Advanced Computer Science and Applications, 9(12), pp.263-267.